8 Kommunikation

Ein Embedded System muss vielfach Daten austauschen, sei es innerhalb des Systems, mit einem anderen Mikrocontroller oder auch mit einem externen Computer (PC, Tablet, Smartphone), wofür die serielle Schnittstelle und der USB die beiden gebräuchlichsten Interfaces darstellen, die in diesem Kapitel näher behandelt werden.

8.1 Serielle Schnittstelle

Bei den Arduino-Systemen ist die serielle Schnittstelle von besonderer Bedeutung, weil sie einerseits für die Programmierung (Abschnitt 7.6) des Mikrocontrollers und anderseits für den Serial Monitor (Abschnitt 7.3.2) zur Datenanzeige verwendet wird, was bei den meisten Arduino-Boards nicht gleichzeitig möglich ist, weil es dort nur einen UART gibt, der die serielle Schnittstelle steuert.

Beim Leonardo- und dem Esplora-Board befindet sich das USB-Interface, welches sonst die UART-Schnittstelle des Controllers verwendet, mit im Chip selbst, so dass die serielle Schnittstelle dort frei bleibt. Bei den Mega-Boards (Abschnitt 4.4) gibt es sogar vier UART-Schnittstellen, so dass dort drei für die universelle serielle Ein- und Ausgabe zu anderen Systemen verwendet werden können. Eine richtige serielle Schnittstelle nach dem RS232-Standard bietet diese Boards alle nicht. Nur das Galileo-Board (Abschnitt 6.3) besitzt einen entsprechenden Treiberchip damit die TTL-Pegel auf den ±12-V-Pegel laut RS232 umgesetzt werden, um mit entsprechenden Geräten kommunizieren zu können. Gleichwohl ist es im Bedarfsfall kein Problem einen RS232-Treiber per Shield oder im Selbstbau nachzurüsten.

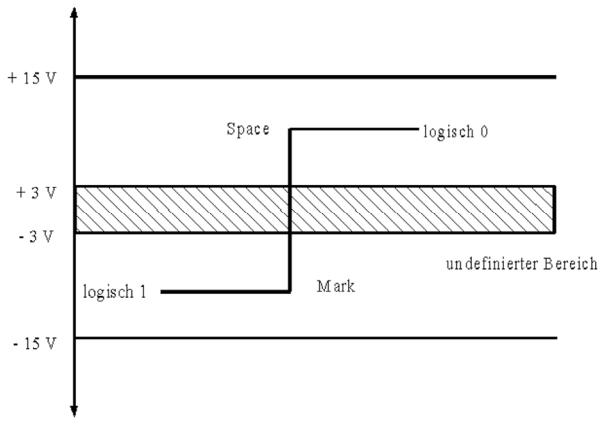


Abbildung 8.1: Signalpegel bei der RS232-Schnittstelle

Solange die serielle Schnittstelle einer Arduino-Platine nicht nach außen zur Gerätekommunikation verwendet wird, sondern nur auf Board-Ebene mit einem anderen Chip anhand des TTL-Pegels arbeiten soll, ist auch keine Signalumsetzung notwendig und es muss lediglich beachtet werden, dass beide Kommunikationspartner die gleichen Übertragungsparameter (siehe Abschnitt 8.1.1) verwenden.

Ein Beispiel ist hierfür im Abschnitt 5.6.1 gezeigt, wo ein XBee-Funkmodul an die UART-Schnitstelle angeschlossen ist. Außerdem funktioniert die UART-USB-Umsetzung für die Kommunikation mit dem PC für den Upload und den Serial Monitor ebenfalls auf TTL-Niveau.

8.1.1 Verdrahtung und Übertragungsparameter

Die serielle Schnittstelle ist grundsätzlich für den Datenverkehr in beiden Richtungen (Duplex) vorgesehen. Die Arduino-UART-Schnittstelle bietet hierfür nur die Minimalausführung einer seriellen Schnittstelle mit einer Transmit- (TXD) und einer Receive-Leitung (RXD) plus einer Masseverbindung (Gnd).

Leitungen zur Steuerung der Datenübername (Handshake) und für die Erhöhung der Datensicherheit mit Bereitschaftsleitungen, wie sie beispielsweise bei den COM-Schnittstellen für Computer (Tabelle 8.1, Abbildung 8.2) üblich sind, gibt es hier nicht. Dies kann bei der Kommunikation mit Geräten, die über eine RS232-Schnittstelle verfügen, durchaus zu Problemen führen, weil diese möglicherweise entsprechende Steuersignale erwarten, so dass entsprechende Brücken (RTS auf CTS, DSR auf DTR) im Anschluss zu löten sind.

Tabelle 8.1: Anschlüsse und Signale der seriellen Schnittstelle für die üblichen Verbindungen

Kürzel	Bezeichnung	Kontakt	1:1	Nullmodem	Ausführung
DCD	Data Carrier Detect	1	-	-	optional
RXD	Receive Data	2	2	3	minimal
TXD	Transmit Data	3	3	2	minimal
DTR	Data Terminal Ready	4	4	6	mit Bereitschaftssignal
GND	Ground, Masse	5	5	5	minimal
DSR	Data Set Ready	6	6	4	mit Bereitschaftssignal
RTS	Request To Send	7	7	8	mit Handshake
CTS	Clear To Send	8	8	7	mit Handshake
RI	Ring Indicator	9	-	-	optional

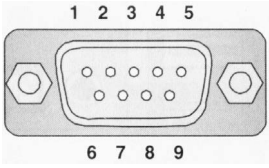


Abbildung 8.2: Der 9-polige Anschluss der seriellen Schnittstelle am PC ist mit Steckkontakten versehen.

Die serielle Kabelverbindung zwischen zwei Geräten wird entweder 1:1 ausgeführt (RXD an RXD, TXD an TXD, GND an GND) oder aber die Sendeleitung des einen Geräts ist die Empfangsleitung des anderen und umgekehrt. Die Verbindungsart mit überkreuz laufenden Signalleitungen wird als *Nullmodem* bezeichnet und typischerweise zwischen

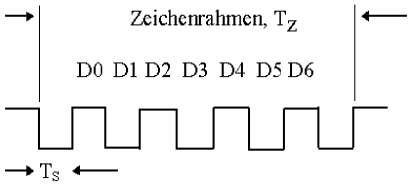
zwei Computern ausgeführt, während die 1:1-Verbindung zwischen einem Computer und einer Peripherieeinheit (Modem, daher die Bezeichnung Nullmodem) üblich ist.

Die RS232-Spezifikation ist recht umfassend und die typische PC-Implementierung ist auch nur eine Untermenge davon, was deshalb zu zahlreichen Varianten und speziell angepassten Adapterkabeln führt. Generell wird auch noch eine synchrone und eine asynchrone Betriebsart unterschieden, wobei letztere das für Computer übliche Verfahren ist, bei dem zur Synchronisation keine separaten Taktleitungen verwendet werden, sondern stattdessen Synchronisationszeichen mit in die zu übertragende Information eingefügt werden. Die für die serielle, asynchrone Übertragung festzulegenden Parameter sind die folgenden, die beim Sender und beim Empfänger stets identisch sein müssen:

- Baudrate
- Startbit
- Stoppbit
- Anzahl der Datenbits
- Parität

Die Baudrate kennzeichnet die Anzahl der Signalzustände, die pro Zeiteinheit übertragen werden, und wird auch als *Schrittgeschwindigkeit* bezeichnet. Sie wird mit der Einheit *Baud* spezifiziert. Übliche Werte hierfür sind:
50 75 110 300 600 1200 2400 4800 9600 19200 38400

Das Startbit wird dem zu übertragenden Zeichen vorangestellt und ist aktiv Low (0). Die Dateninformation besteht in der Regel aus sieben Datenbits (ASCII), wobei das LSB zuerst übertragen wird. Das Ende des Zeichens wird meist durch ein oder zwei Stopp-Bits, welche aktiv High (1) sind, gekennzeichnet. Tritt während der Übertragung eine Pause ein, wird eine 1 (Mark) eingefügt. Zur Fehlererkennung kann ein Paritätsbit mit übertragen werden.



T_S = Schrittdauer
ST = Startbit
P = Paritätsbit
SP = Stoppbit(hier für gerade Parität)
T_Z = Übertragungsdauer eines Zeilenrahmens

Abbildung 8.3: Beispiel für eine asynchrone Übertragung des ASCII-Zeichens »U« (55h) mit gerader Parität

Die Paritätsprüfung (Parity Check) dient generell dazu, einzelne fehlerhafte Übertragungsbits erkennen zu können, dabei existieren die gerade (even) und die ungerade Parität (odd). Ist zwischen Sender und Empfänger eine gerade Parität vereinbart worden, zählt der Datensender alle Bits, die den Wert 1 haben, und setzt das Paritätsbit auf 0, wenn die

Summe der Bits gerade ist. Das Paritätsbit wird auf 1 gesetzt, wenn die Summe der Bits ungerade ist. Der Empfänger zählt alle Bits, die eine 1 haben, und das Paritätsbit. Ist die Quersumme ungerade, liegt ein Übertragungsfehler vor.

Tabelle 8.2: Paritätsprüfung mit Paritätsbit

Bitfolge ohne Paritätsbit	Bitfolge mit Paritätsbit gerade Parität	Bitfolge mit Paritätsbit ungerade Parität
0101110	01011100	01011101
1101110	11011101	11011100

Das Verfahren für die ungerade Paritätsprüfung entspricht dem der geraden Paritätsprüfung, lediglich die Summe aus den gesetzten Datenbits und dem Paritätsbit muss immer ungerade sein. Mit einem Paritätsbit ist keine Fehlerkorrektur möglich, sondern es kann lediglich ein Fehler als solcher erkannt werden. Die Paritätsprüfung eignet sich daher eher für kurze und wenig störanfällige Übertragungswege.

In den Computereinstellungen für die seriellen Schnittstellen ist noch die Option der Flusssteuerung üblich. Sie ist dann auf *Hardware* zu schalten, wenn die Steuerung anhand der (verdrahteten) Handshake- und Betriebsbereitschaftsleitungen erfolgt. Das Xon/Xoff-Protokoll regelt die Datenübernahme stattdessen per Software.

8.1.2 Implementierung und Anwendung

Die serielle Schnittstelle wird bei den Arduino-Controllern intern mit einer USART-Einheit (**U**niversal **S**ynchronous and **A**synchronous serial **R**eceiver and **T**ransmitter) realisiert, so dass prinzipiell auch die synchrone Betriebsart möglich ist, die der USART (nur) für die SPI-Schnittstelle einsetzt. Der USART (Abbildung 8.4) ist verhältnismäßig komplex und verfügt über fünf Register: Data I/O- (UDRn), Baudrate- (UBRRn) und drei Control- und Status-Register (UCSRn A/B/C).

Arduino-gemäß wird diese Komplexität meist nicht offenbar, weil es standardmäßig die Serial-Funktionen wie beispielsweise *Serial.begin ()* und *Serial.print()* gibt, die direkt auf die Hardware USARTs wirken. Außerdem ist noch die einfach einzusetzende *Software Serial Library* verfügbar mit der (beliebige) Digital-Ports mit einer seriellen Funktionalität ausgestattet werden können.

Tabelle 8.3: Befehle für serielle Schnittstellen

Befehl	Funktion
SoftwareSerial(rxPin, txPin)	erstellt ein neues Objekt (Serial)
Serial.available()	Anzahl der empfangenen Bytes, die anstehen und gelesen werden können
Serial.begin(speed)	Setzen der Baudrate
Serial.isListening()	Test, ob die serielle Schnittstelle auf eingehende Daten reagieren kann
Serial.Listen()	den Port Serial aktivieren
Serial.Overflow()	Test, ob ein Überlauf im Serial-Buffer stattgefunden hat
Serial.print(data)	Daten auf den TX-Pin schreiben
Serial.println(data)	Daten auf den TX-Pin schreiben, gefolgt von CURSOR ZURÜCK und NEUE ZEILE
Serial.read()	liefert ein Zeichen, das auf dem RX-Pin eingetroffen ist
Serial.write(data)	unformatierte Daten (Raw Bytes) auf den TX-Pin schreiben

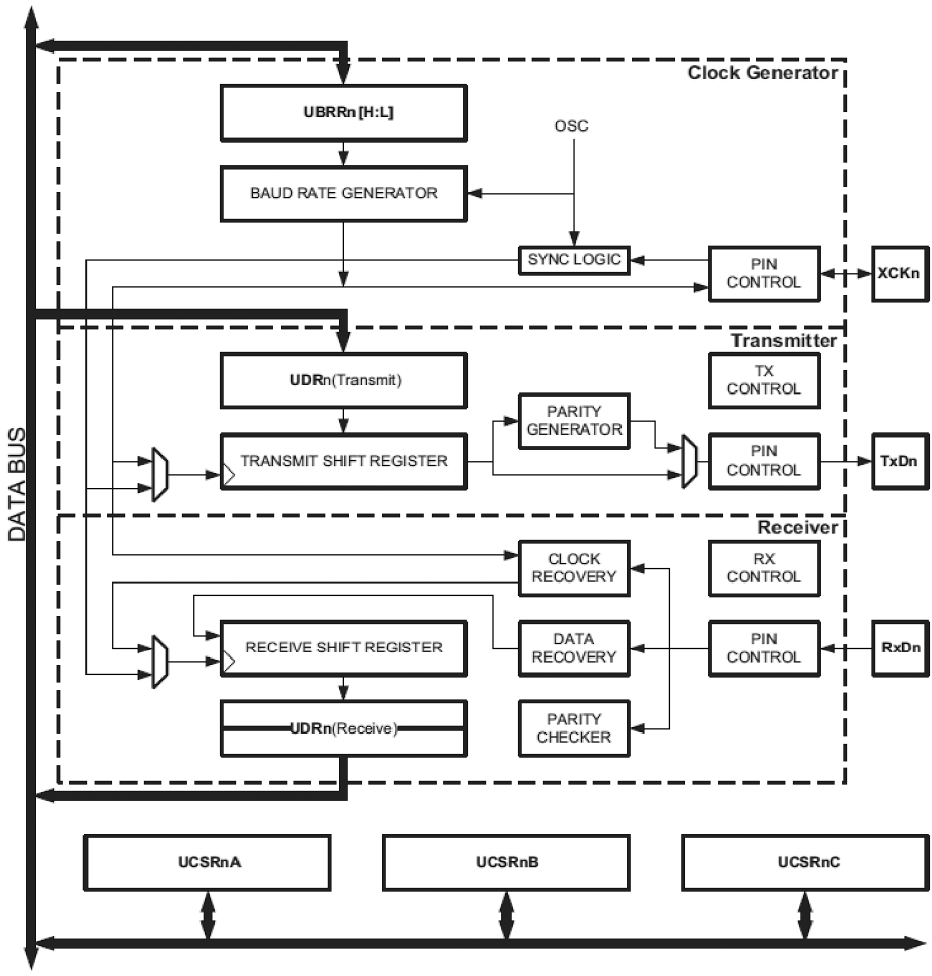


Abbildung 8.4: Der USART im ATMega168/328

Die Tabelle 8.3. zeigt eine Übersicht mit einer kurzen Befehlserläuterung. Auch wenn ein Arduino-Board (wie die Mega-Typen) über mehrere serielle Ports verfügt, kann zur gleichen Zeit immer nur von einem gelesen werden, was auch für die Software Serial Ports gilt. Eine Umschaltung zwischen mehreren Ports kann mithilfe der Listen-Funktion erfolgen, wie es im Listing gezeigt ist.

Listing: Verwendung von USART und zwei Software-Serial-Ports

```
#include <SoftwareSerial.h>

// Software Serial : TX = digital pin 10, RX = digital pin 11
SoftwareSerial portOne(10, 11);

// Software Serial : TX = digital pin 8, RX = digital pin 9
SoftwareSerial portTwo(8, 9);

void setup()
{
```

```
  // Starten des Hardware Serial Ports
  Serial.begin(9600);

  // Starten von zwei Software Serial Ports
  portOne.begin(9600);
  portTwo.begin(9600);
}

void loop()
{
  portOne.listen();


     if (portOne.isListening())
        {
          Serial.println("Port One is listening!");
        }
     else
        {
          Serial.println("Port One is not listening!");
        }

     if (portTwo.isListening())
        {
         Serial.println("Port Two is listening!");
        }
     else
        {
         Serial.println("Port Two is not listening!");
        }
}
```

Mit dem Serial Monitor ist nicht nur eine Datenausgabe möglich, sondern es können dort auch Daten eingegeben und an den USART geschickt werden, was bedeutet, dass diese Daten dann prinzipiell auch von einem PC oder einem anderem Gerät aus gesendet werden und vom folgenden Sketch empfangen werden könnten.

Listing: Einfacher serieller Datenempfang

```
int  serIn; // Variable für gelesene Bytes

void setup()
{
  Serial.begin(9600);

}

 void loop ()
{
  if(Serial.available())
    {
     Serial.print("Datenemfang: ");

     while (Serial.available()>0)
           {
            serIn = Serial.read();        // Lesen vom Serial Port
            Serial.print(serIn, BYTE);    // Ausgabe der Zeichen
           }

     Serial.println();
     }

  delay(1000); // Verzögerung für Ausgabe
}
```